

# JOB SHEET 9

## Pointer

*"Everybody should learn to program a computer,  
because it teaches you how to think."*

### TUJUAN PEMBELAJARAN

1. Mampu menjelaskan dan mengimplementasikan *Pointer* dalam pemrograman menggunakan IDE.

### POKOK MATERI

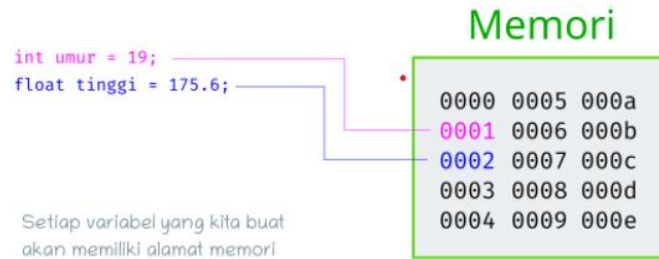
1. Pengertian Pointer
2. Penggunaan Pointer
3. Pointer untuk pass by reference
4. Pointer untuk mengakses array

### URAIAN MATERI

#### **A. Pengertian Pointer**

Setiap variabel yang kita buat pada program akan memiliki alamat memori. Alamat memori berfungsi untuk menentukan lokasi penyimpanan data pada memori (RAM). Kadang alamat memori ini disebut reference atau referensi.

Coba perhatikan gambar ini:



Pada gambar ini, kita membuat dua variabel. yakni umur dan tinggi. Kedua variabel ini punya alamat memori masing-masing. Variabel umur alamat memorinya adalah 0001, sedangkan tinggi alamat memorinya 0002. Begitu seterusnya.

Pokoknya, setiap kita membuat variabel pasti akan punya alamat memori. Untuk melihat alamat memori yang digunakan pada variabel, kita bisa pakai simbol `&` (emphasis).

Contoh:

```
#include <stdio.h>

void main () {

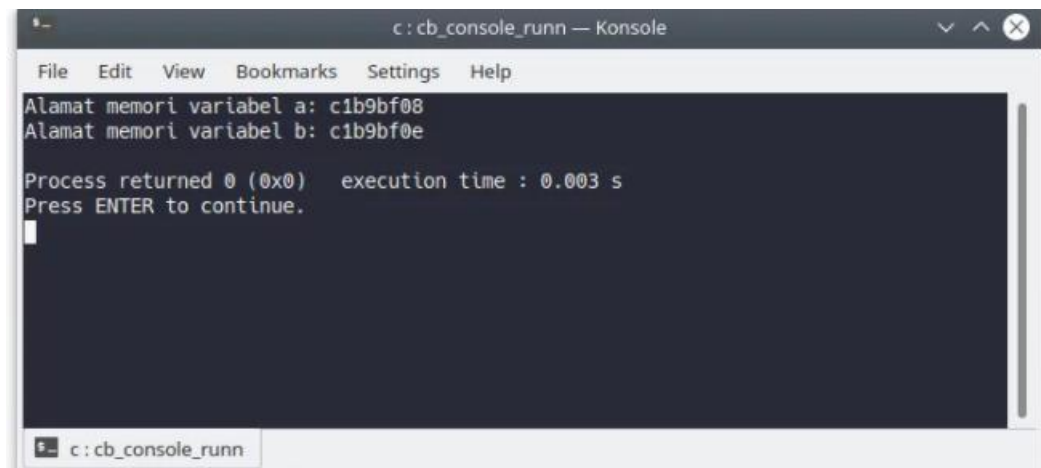
    int a;
    char b[10];

    printf("Alamat memori variabel a: %x\n", &a);
    printf("Alamat memori variabel b: %x\n", &b);

}
```

Pada program tersebut, kita menggunakan simbol `&` untuk mengambil alamat memori dari variabel `a` dan `b`. Lalu menggunakan format specifier `%x` untuk menampilkannya dalam bilangan heksadesimal.

Hasil outputnya:



```
c: cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
Alamat memori variabel a: c1b9bf08
Alamat memori variabel b: c1b9bf0e
Process returned 0 (0x0) execution time : 0.003 s
Press ENTER to continue.
```

Lalu apa hubungannya alamat memori dengan pointer?

Pointer adalah sebuah variabel yang berisi alamat memori dari variabel yang lain. Pointer nantinya akan bisa mengakses data yang ada di suatu alamat memori.

Kata kunci yang perlu kamu ingat:

**"Pointer berisi alamat memori"**

### Penggunaan Pointer

Pointer dibuat dengan menambahkan simbol \* (asterik) di depan namanya, kemudian diisi dengan alamat memori yang akan digunakan sebagai referensi.

Contoh:

```
int *pointer1 = 00001;
```

Maka \*pointer1 akan bisa mengakses data yang ada pada alamat memori 00001. Dengan kata lain, si \*pointer1 akan menggunakan alamat 00001 sebagai referensinya.

Kita juga bisa membuat pointer, tanpa harus mengisinya langsung dengan alamat memori.

Contoh:

```
int *pointer_ku;

// atau bisa juga

int *pointer_ku = NULL;
```

Maka `*pointer_ku` akan menggunakan alamat memori 00000, alamat memori ini khusus untuk menyimpan data null atau data kosong.

Sekarang masalahnya:

Karena kita tidak bisa lihat daftar alamat memori secara langsung, kita akan kesulitan memberikan referensi alamat memori kepada pointer.

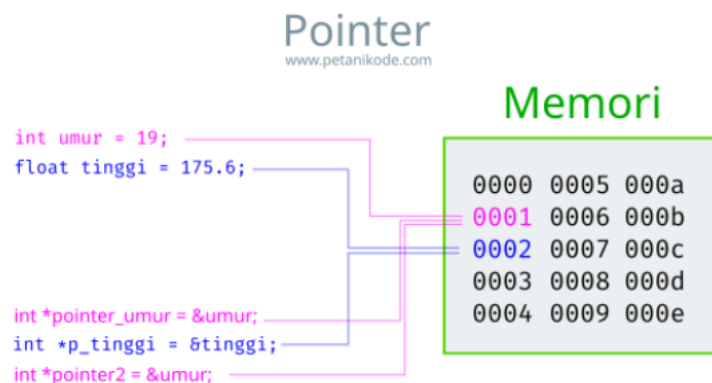
Belum lagi, beda komputer beda juga alamat memorinya dan alamat memori akan selalu berubah setiap program dieksekusi.

Solusinya:

Kita harus mengambil alamat memori dari variabel yang lain.

Masih ingat caranya? Ya, dengan menggunakan simbol `&`.

Coba perhatikan gambar ini:



Pada gambar ilustrasi ini, kita membuat tiga pointer yakni `*pointer_umur`, `*p_tinggi`, dan `*pointer2`. Pointer `*pointer_umur` dan `*pointer2` menggunakan referensi alamat memori 0001. Alamat memori ini merupakan alamat memori dari variabel umur. Lalu, si pointer `p_tinggi` menggunakan alamat memori dari

variabel tinggi. Dengan begini. Pointer akan bisa mengakses data yang tersimpan di dalam alamat memori tersebut.

Buatlah program baru dengan nama contoh\_pointer.c, kemudian isi dengan kode berikut:

```
#include <stdio.h>

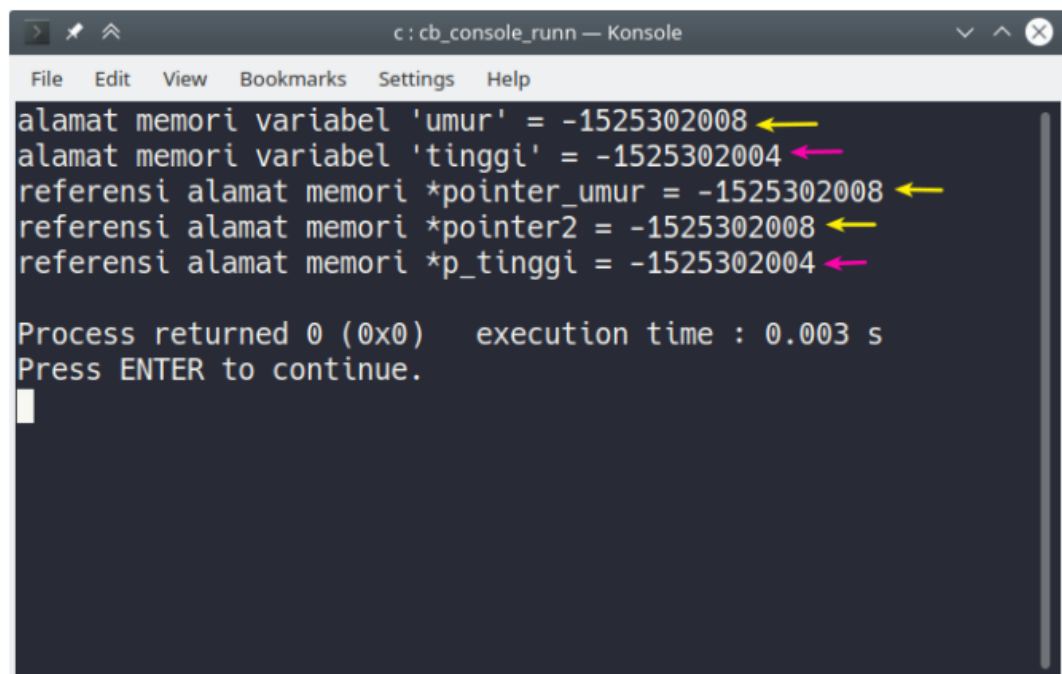
void main() {
    // membuat variabel
    int umur = 19;
    float tinggi = 175.6;

    // membuat pointer
    int *pointer_umur = &umur;
    int *pointer2 = &umur;
    float *p_tinggi = &tinggi;

    // mencetak alamat memori variabel
    printf("alamat memori variabel 'umur' = %d\n", &umur);
    printf("alamat memori variabel 'tinggi' = %d\n", &tinggi);
    // mencetak referensi alamat memori pointer
    printf("referensi alamat memori *pointer_umur = %d\n", pointer_umur);
    printf("referensi alamat memori *pointer2 = %d\n", pointer2);
    printf("referensi alamat memori *p_tinggi = %d\n", p_tinggi);
}
```

Setelah itu, coba compile dan jalankan.

Hasil outputnya:



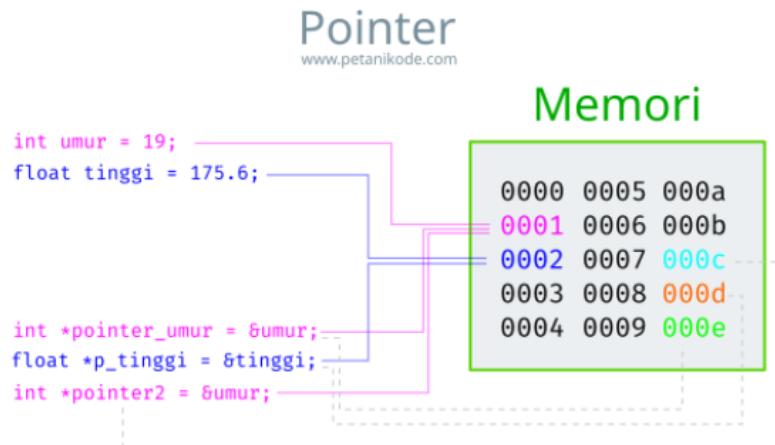
```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help

alamat memori variabel 'umur' = -1525302008
alamat memori variabel 'tinggi' = -1525302004
referensi alamat memori *pointer_umur = -1525302008
referensi alamat memori *pointer2 = -1525302008
referensi alamat memori *p_tinggi = -1525302004

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

Alamat memori yang digunakan sebagai referensi pada pointer akan sama dengan alamat memori dari variabel yang kita pakai sebagai referensi. pointer juga punya alamat memorinya sendiri.

Coba perhatikan gambar ini:



Sama seperti variabel biasa. Jika ingin melihat alamat memori dari pointer, maka kita harus menggunakan simbol &.

Contoh:

```

#include <stdio.h>

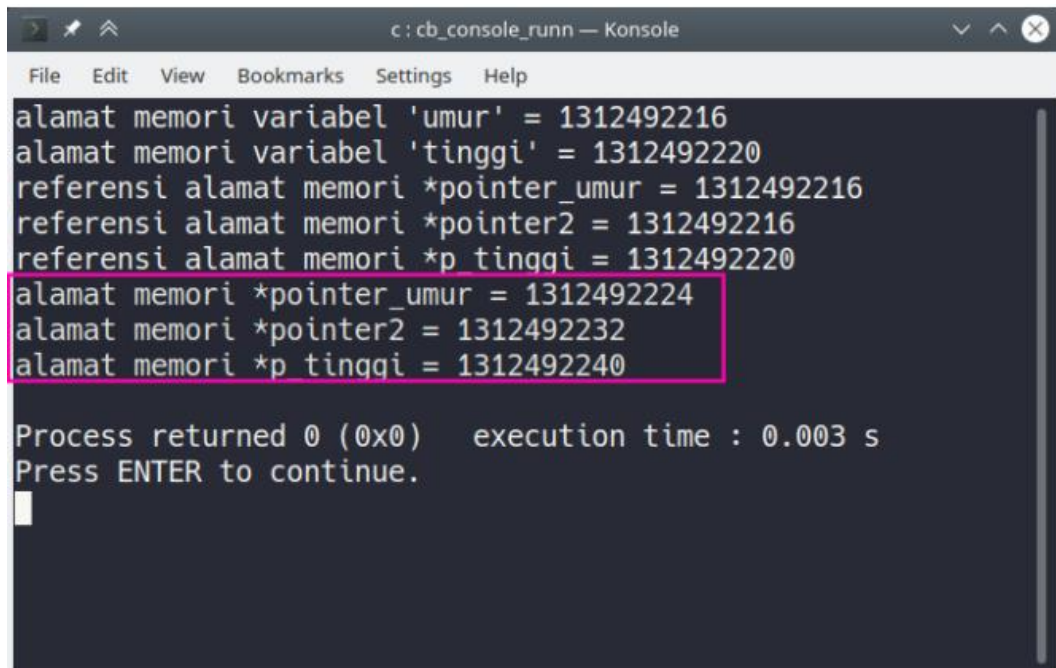
void main(){
    // membuat variabel
    int umur = 19;
    float tinggi = 175.6;

    // membuat pointer
    int *pointer_umur = &umur;
    int *pointer2 = &umur;
    float *p_tinggi = &tinggi;

    // mencetak alamat memori variabel
    printf("alamat memori variabel 'umur' = %d\n", &umur);
    printf("alamat memori variabel 'tinggi' = %d\n", &tinggi);
    // mencetak referensi alamat memori pointer
    printf("referensi alamat memori *pointer_umur = %d\n", pointer_umur);
    printf("referensi alamat memori *pointer2 = %d\n", pointer2);
    printf("referensi alamat memori *p_tinggi = %d\n", p_tinggi);

    // mencetak alamat memori pointer
    printf("alamat memori *pointer_umur = %d\n", &pointer_umur);
    printf("alamat memori *pointer2 = %d\n", &pointer2);
    printf("alamat memori *p_tinggi = %d\n", &p_tinggi);
}
    
```

Hasilnya:



```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
alamat memori variabel 'umur' = 1312492216
alamat memori variabel 'tinggi' = 1312492220
referensi alamat memori *pointer_umur = 1312492216
referensi alamat memori *pointer2 = 1312492216
referensi alamat memori *p tinggi = 1312492220
alamat memori *pointer_umur = 1312492224
alamat memori *pointer2 = 1312492232
alamat memori *p tinggi = 1312492240

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

Apa bedanya alamat memori pointer dengan alamat referensi?

Alamat memori pointer adalah alamat memori yang digunakan untuk menyimpan pointer. Sedangkan alamat referensi adalah alamat yang akan menjadi referensi dari pointer.

Ingat:

"Pointer akan bisa mengakses isi data pada alamat referensi yang diberikannya"

Jika kita bisa menggunakan variabel biasa, ngapain pakai pointer?

Penggunaan pointer sebenarnya opsional, kamu boleh pakai. boleh juga tidak.

Namun. Pada kondisi tertentu, penggunaan pointer lebih optimal.

## Mengakses data dengan Pointer

Contoh:

```
#include <stdio.h>

void main(){
    // membuat variabel score
    int score = 0;

    // membuat pointer dan referensikan dengan alamat
    // dari variabel score
    int *p_score = &score;

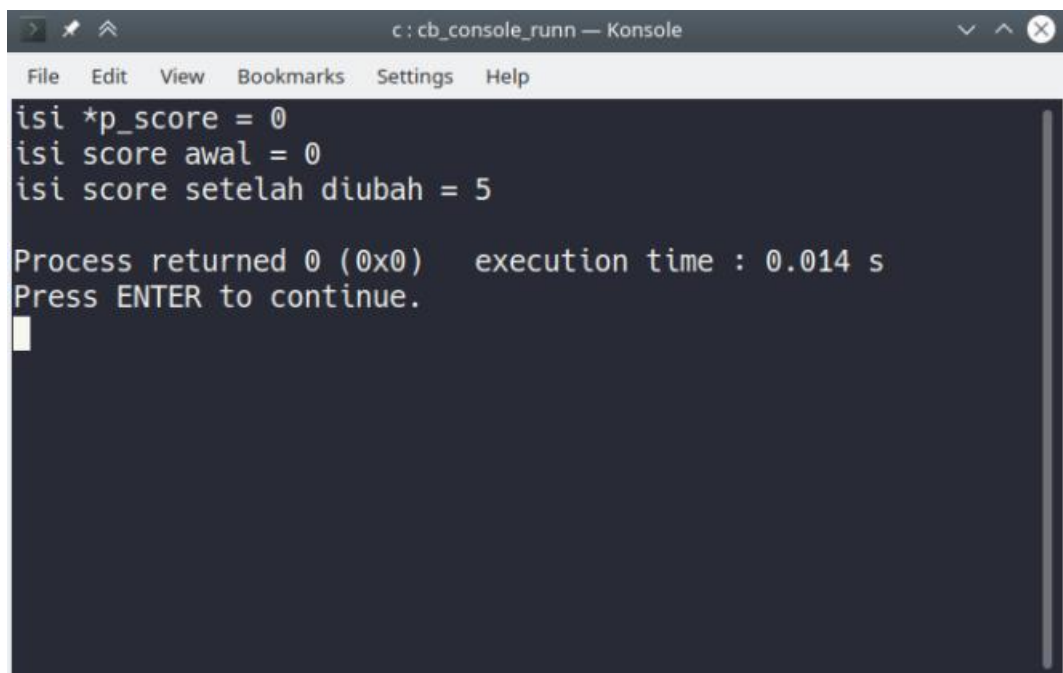
    // mengakses data variabel score dari pointer
    printf("isi *p_score = %d\n", *p_score);

    printf("isi score awal = %d\n", score);

    // mengubah data variabel score dari pointer
    *p_score = 5;

    // melihat isi variabel score
    printf("isi score setelah diubah = %d\n", score);
}
```

Hasilnya:



```
c:\cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
isi *p_score = 0
isi score awal = 0
isi score setelah diubah = 5

Process returned 0 (0x0)   execution time : 0.014 s
Press ENTER to continue.
```

Isi variabel score akan berubah, karena kita mengubahnya dari pointer.



Kapan harus menggunakan pointer? kita tidak harus selalu menggunakan pointer dalam program. Namun, ada beberapa kasus tertentu yang menyarankan menggunakan pointer daripada cara biasa. Karena terbukti, dengan pointer performa program akan lebih optimal.

Mengapa pointer diciptakan?

Jadi zaman dulu, memori komputer itu sangat terbatas. Saat mengelola struktur data kompleks seperti data pada array, linked list, tree, dan sebagainya. sering kali memakan banyak memori. Oleh sebab itu, diciptakanlah pointer agar mudah membuat struktur data tersebut, dan tentunya lebih hemat memori.

### Studi Kasus: Pointer untuk pass by reference

Petama kita akan coba menggunakan pointer untuk melakukan passing argumen berdasarkan referensinya (pass by reference).

Contoh:

```
#include <stdio.h>

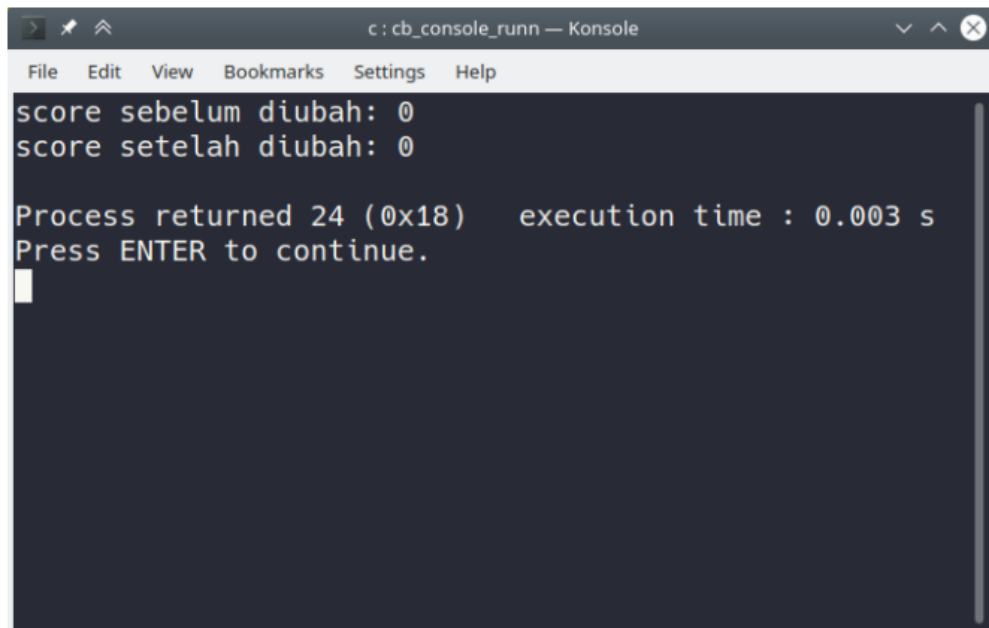
void add_score(int score){
    score = score + 5;
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(score);
    printf("score setelah diubah: %d\n", score);
}
```

Pada program ini, kita membuat fungsi dengan nama `add_score()` untuk menambahkan nilai `score` sebanyak 5.

Tapi ketika dijalankan:



```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
score sebelum diubah: 0
score setelah diubah: 0

Process returned 24 (0x18)   execution time : 0.003 s
Press ENTER to continue.
```

Nilai variabel score tidak berubah, ia tetap bernilai 0.

Mengapa?

Ini karena kita melakukan pass by value, bukan pass by reference.

Variabel score kan dibuat di dalam fungsi main(), lalu ketika fungsi add\_score() mencoba mengubah nilainya. maka perubahan hanya terjadi secara lokal di dalam fungsi add\_score() saja.

Coba buktikan dengan mengubah fungsi add\_score() menjadi seperti ini:

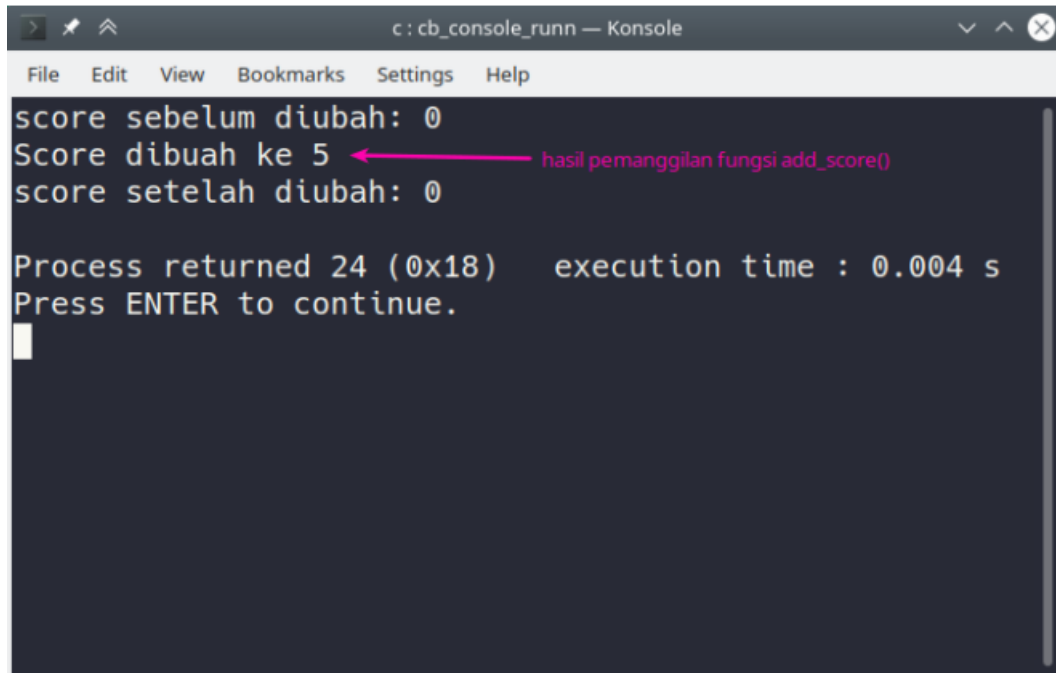
```
#include <stdio.h>

void add_score(int score){
    score = score + 5;
    printf("Score diubah ke %d\n", score);
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(score);
    printf("score setelah diubah: %d\n", score);
}
```

Hasilnya:



```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
score sebelum diubah: 0
Score dibuah ke 5 ← hasil pemanggilan fungsi add_score()
score setelah diubah: 0

Process returned 24 (0x18)   execution time : 0.004 s
Press ENTER to continue.
```

Nilai score pada fungsi `add_score()` sudah berubah menjadi 5, namun variabel `score` pada fungsi `main()` akan tetap bernilai 0. Di sinilah kita harus menggunakan pointer untuk melakukan pass-by-reference.

Sekarang, coba ubah kode programnya menjadi seperti ini:

```
#include <stdio.h>

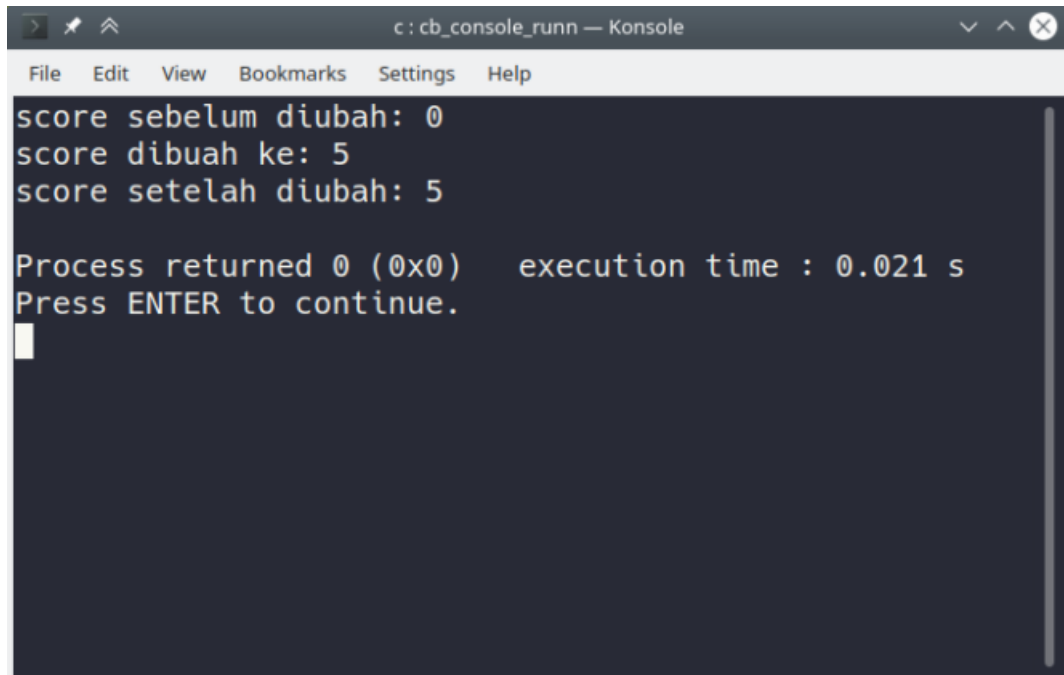
void add_score(int *score){
    *score = *score + 5;
    printf("score dibuah ke: %d\n", *score);
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(&score);
    printf("score setelah diubah: %d\n", score);
}
```

Karena argumen fungsi `add_score()` kita ubah menjadi pointer, maka kita harus memberikan alamat memori saat memanggilmnya.

Maka hasilnya:



```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
score sebelum diubah: 0
score dibuah ke: 5
score setelah diubah: 5

Process returned 0 (0x0)   execution time : 0.021 s
Press ENTER to continue.
```

Setiap fungsi `add_score()` dipanggil atau dieksekusi, maka nilai variabel `score` akan bertambah 5.

Coba kita ubah menjadi seperti ini:

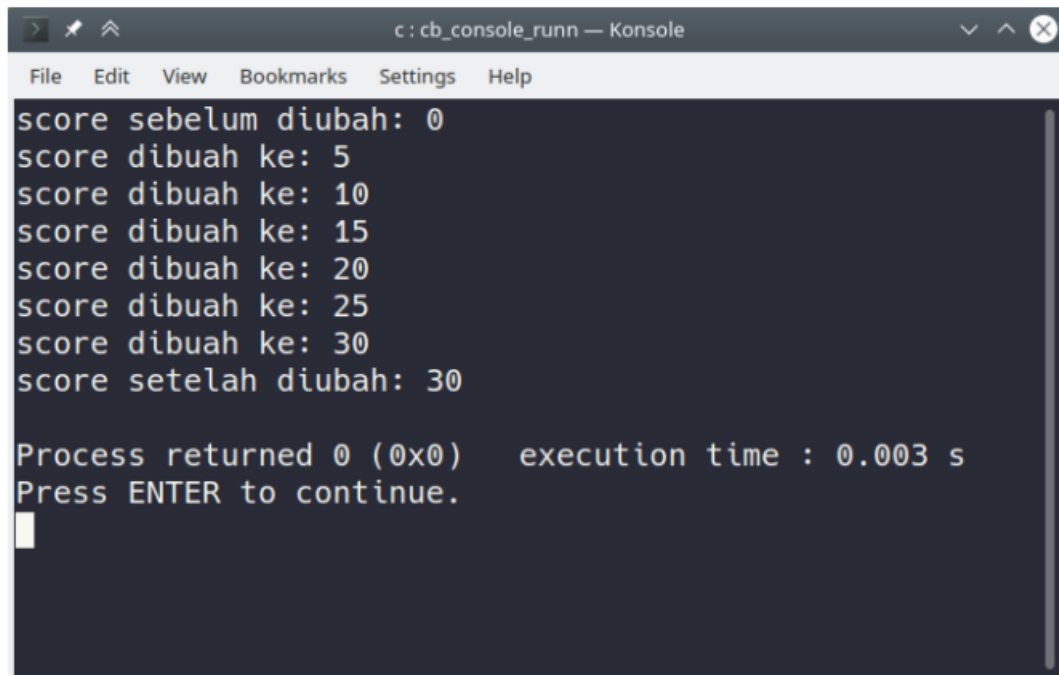
```
#include <stdio.h>

void add_score(int *score){
    *score = *score + 5;
    printf("score dibuah ke: %d\n", *score);
}

void main(){
    int score = 0;

    printf("score sebelum diubah: %d\n", score);
    add_score(&score);
    add_score(&score);
    add_score(&score);
    add_score(&score);
    add_score(&score);
    add_score(&score);
    printf("score setelah diubah: %d\n", score);
}
```

Hasilnya:



```
c : cb_console_runn — Konsole
File Edit View Bookmarks Settings Help
score sebelum diubah: 0
score dibuah ke: 5
score dibuah ke: 10
score dibuah ke: 15
score dibuah ke: 20
score dibuah ke: 25
score dibuah ke: 30
score setelah diubah: 30

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.

```

### Studi Kasus: Pointer untuk Mengakses Array

Pointer juga sering digunakan untuk mengakses elemen array.

Contoh: pointer\_array.c

```
#include <stdio.h>

void main(){
    printf("## Program Antrian CS ##\n");

    char no_antrian[5] = {'A', 'B', 'C', 'D', 'E'};

    // menggunakan pointer
    char *ptr_current = &no_antrian;

    for(int i = 0; i < 5; i++){
        printf("👤 Pelanggan dengan no antrian %c silahkan ke loket!\n", *ptr_current);
        printf("Saat ini CS sedang melayani: %c\n", *ptr_current);
        printf("----- Tekan Enter untuk Next -----");
        getchar();
        ptr_current++;
    }

    printf("✅ Selesai");
}
```

Pada program ini, kita menggunakan `ptr_current` untuk mengakses elemen array. Saat pertama kali dibuat, pointer `ptr_current` akan mereferensi pada elemen pertama array. Lalu pada perulangan dilakukan increment `ptr_current++`, maka pointer ini akan mereferensi ke elemen array selanjutnya.

Hasilnya:

```
## Program Antrian CS ##
▶ Pelanggan dengan no antrian A silahkan ke loket!
Saat ini CS sedang melayani: A
----- Tekan Enter untuk Next -----
▶ Pelanggan dengan no antrian B silahkan ke loket!
Saat ini CS sedang melayani: B
----- Tekan Enter untuk Next -----
▶ Pelanggan dengan no antrian C silahkan ke loket!
Saat ini CS sedang melayani: C
----- Tekan Enter untuk Next -----
▶ Pelanggan dengan no antrian D silahkan ke loket!
Saat ini CS sedang melayani: D
----- Tekan Enter untuk Next -----
```

## LATIHAN

1. Buat program dengan menggunakan pointer sehingga menghasilkan keluaran sebagai berikut:

D

ND

AND

LAND

RLAND

ORLAND

BORLAND

2. Buat program untuk menampilkan kalimat terbalik dari suatu kalimat yang diinputkan. Misal:

Kalimat Masukan : HELLO

Kebalikannya : OLLEH

3. Buatlah program untuk menjawab pertanyaan berikut:

A. Lesley = 57082

Layla = Lesley

Balmond = Layla + 1

- a) Berapakah nilai Layla?
- b) Berapakah nilai Balmond?

B. Lesley = 57082

Layla = &Lesley

Balmond = \*Layla + 1

- a) Berapakah nilai Layla?
- b) Berapakah nilai Balmond?

## REFERENCES

---

1. The C Programming Language. 2nd Edition
2. [https://en.wikipedia.org/wiki/C\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/C_(programming_language))
3. [https://en.wikipedia.org/wiki/Imperative\\_programming](https://en.wikipedia.org/wiki/Imperative_programming)
4. <https://www.petanikode.com/tutorial/c/>
5. <https://www.learn-c.org/>
6. <https://www.tutorialspoint.com/cprogramming/index.htm>
7. <https://www.programiz.com/>
8. <https://www.dicoding.com/>
9. <https://data-flair.training/blogs/c-tutorials-home/>